Abstracting the Indistinguishable in ASP

Zeynep G. Saribatur[1] , **Markus Hecher**[2] , **Johannes K. Fichte**[3]

[1]Institute of Logic and Computation, TU Wien
[2]University of Potsdam, Germany & University of Artois, CNRS, UMR8188 (CRIL), France
[3]Department of Computer and Information Science (IDA), Linköping University
zeynep.saribatur@tuwien.ac.at, hecher@cril.fr, johannes.fichte@liu.se

Abstract

Answer Set Programming (ASP) is a popular knowledge representation and reasoning framework with countless applications for modeling and solving combinatorial problems. With increasingly large applications, identifying crucial details and *collapsing irrelevant* or *indistinguishable* information becomes crucial for a human in the loop. This idea led to the investigation of different notions of abstraction, which relate to forgetting and projection. Very recently, clustering formalisms have been designed that also preserve program dependencies. However, crucial computational properties, such as finding abstractions, remained entirely unexplored to date. In this work, we examine the computational complexity of existing abstraction techniques based on clustering (faithful and uniform abstractions). Besides checking, we tackle the crucial question of *constructing abstractions*, by also proposing a novel syntactic operator to achieve uniform abstractions, when possible. Moreover, we investigate whether symmetry detection can be used to determine abstractable parts, and give insights on the difference of interchangeability and indistinguishability, by introducing a relaxation of the uniform abstraction condition to capture semantic indistinguishability. We observe that this notion enables us to obtain intuitive faithful abstractions, while symmetry does not. We explore properties needed to reach abstractions from syntactic symmetry.

1 Introduction

Answer set programming (ASP) is a widely used declarative programming framework where programs consist of rules and constraints describing solutions, so-called *answer sets* [Brewka *et al.*, 2011]. Today, we see numerous applications of ASP in artificial intelligence, automated reasoning, and symbolic planning thanks to highly capable solvers [Gebser *et al.*, 2014; Alviano *et al.*, 2015; Alviano *et al.*, 2017]. With increasingly large applications, generalizing from smaller details becomes crucial as it allows humans to identify the problem structure and the essential details. Constructing "good" abstractions over the problems from a few instances is key to finding generalized solutions that work for all potential instances with similar structure. The obtained generalized solutions can also be used as a guide in searching for a solution to the original problem.

Humans naturally and intuitively abstract, but defining notations and designing automated techniques turn out to be extremely challenging. While approaches that involve forgetting [Gonçalves *et al.*, 2023; Doherty and Szałas, 2024], equivalence-based rewriting [Pearce, 2004; Eiter *et al.*, 2004], partial evaluations [Brass and Dix, 1997; Janhunen *et al.*, 2006] and dependency preserving abstractions [Saribatur and Woltran, 2023; Saribatur *et al.*, 2024] show potential in obtaining simplifications of programs that might distinguish the essential details, the computational complexity of *coming up* with a simplified or abstracted program has received little attention so far.

Using the traditional notions of strong and uniform equivalence, which preserve answer sets between programs when they are extended with further programs and facts, respectively, easily result in quite technical constructions of the simplifications [Pearce, 2004; Eiter *et al.*, 2004]. Furthermore an abstraction over the vocabulary cannot be treated. In forgetting [Gonçalves *et al.*, 2023] the vocabulary is reduced by forgetting atoms, while the construction of the simplified programs satisfying the interesting properties of strong- and uniform persistence are also highly technical [Gonçalves *et al.*, 2021; Berthold *et al.*, 2019; Berthold, 2022] which was shown to be due to the aim to preserve dependencies w.r.t. the remaining atoms [Saribatur and Woltran, 2024].

Faithful abstractions [Saribatur *et al.*, 2021] are first to define an abstraction via clustering atoms (in the non-ground setting) in the vocabulary, while preserving the answer sets, which was also defined for omitting atoms [Saribatur and Eiter, 2021]. Though the focus of [Saribatur and Eiter, 2021; Saribatur *et al.*, 2021] was on over-approximating abstractions, and dealing with spurious abstract answer sets, thus constructing faithful abstractions was not investigated. Recently, abstraction by clustering notion was brought in to ground programs and restricted to achieve a uniform equivalence-like property [Saribatur *et al.*, 2024], with the aim to capture irrelevancy of atoms, when the program is further extended with facts, which goes towards capturing generalized reasoning. A syntactic operator was proposed to obtain uniform abstractions, when possible, for certain fragments, and the complexity

of checking whether a program is a uniform abstraction was shown to be in Π_3^P while hardness was left open.

A long open question is whether abstracting over atoms for getting rid of irrelevant details is similar to symmetry detection, especially whether it is possible to abstract over symmetry, c.f. [Fox *et al.*, 2005; Yu *et al.*, 2023]. So far in ASP, symmetry notion has focused on detecting symmetries for constructing symmetry breaking constraints to help with solving [Drescher *et al.*, 2011; Tarzariol *et al.*, 2023]. The potential of these symmetries to obtain an abstracted program was not investigated.

Here, we focus on faithful and uniform abstractions with clustering, take a look at symmetry with a fresh perspective, and show how to obtain intuitive abstractions over the indistinguishable. We systematically analyse the computational complexity of testing abstractions and checking for existence of abstractions. *Our main contributions* are as follows:

1. We examine the computational complexity of checking abstractions and existence of abstractions for established notions (faithful and uniform) using clustering (see Tables 1 and 2). Thereby, we provide novel insights into the crucial question of *existence of abstractions*.
2. We propose a novel syntactic operator that can construct uniform abstractions of programs, if exists. We show that the known necessary condition for uniform abstractions is in fact also *sufficient*, without directly diving into semantic technicalities (e.g., HT models).
3. We investigate the potential of syntactic and semantic symmetries in ASP in determining abstractions. We establish that *symmetry and abstraction notions are orthogonal*, due to the difference of interchangeability and indistinguishability.
4. We define a relaxation of the shown condition for uniform abstractions to capture atoms being semantically indistinguishable, which allows our syntactic operator to obtain *intuitive faithful abstractions*.
5. We show, with further syntactic restrictions, *syntactically symmetric atoms become semantically indistinguishable* and achieve uniform abstractions. We illustrate the usefulness of abstracting such symmetries when compared to breaking them.

Related Works and Boarder Relation to AI. Since the early days of AI research, several works have explored the definition of notions of abstraction for various domains, with some focusing on theoretical properties and others on domain-specific applications [Saitta and Zucker, 2013]. In planning, researchers investigated abstraction notions for generalized planning [Banihashemi *et al.*, 2017; Srivastava *et al.*, 2011; Illanes and McIlraith, 2019; Giacomo *et al.*, 2025], where a plan that can solve multiple instances of a problem is obtained from a given a set of planning instances. Early approaches aimed to ignore irrelevant parts [Nebel *et al.*, 1997] and synthesized invariants aiming to speed up the search [Rintanen, 2000]. Recent approaches aim at handling symmetries [Sievers *et al.*, 2019] and compute abstractions for certain types of planning problems [Dong *et al.*, 2025; Cui *et al.*, 2021]. However, it remains entirely unclear how these approaches can be generalized to other AI formalisms. More importantly, the actual computational cost of finding good abstractions in a domain-independent setting, especially those that provide generalization abilities, remained largely open. Symmetry is widely used constraint solving and propositional satisfiability [Krishnamurthy, 1985; Devriendt *et al.*, 2016] and recently also to lift tasks [Carbonnelle *et al.*, 2024]. Breaking symmetries for faster solving has been addressed in a couple of publications [Benhamou, 2013; Devriendt and Bogaerts, 2016; Tarzariol *et al.*, 2023].

	faithful		uniform	
Strat	**P**	Lem. 7	**coNP-c**	Th. 8
Normal	$\mathbf{\Pi_2^P}$**-c**	Lem. 7	$\mathbf{\Pi_2^P}$	Th. 8
Disj	$\mathbf{\Pi_3^P}$**-c**	Lem. 7	$\mathbf{\Pi_3^P}$**-c***	Th. 8

Table 1: (Testing abstractions): Complexity results of abstracting by clustering. Decide whether Q abstracts P using m, given an original program P, a resulting program Q, and mapping between atoms of P and Q; * indicates known membership [Saribatur *et al.*, 2024].

	faithful / (Q consistent)		uniform	
Strat	triv. / (**P**)	Th. 11 / (12)	$\mathbf{\Sigma_2^P}$	Th. 9
Normal	triv. / (NP-**c**)	Th. 11 / (12)	$\mathbf{\Sigma_3^P}$	Th. 9
Disj	triv. / ($\mathbf{\Sigma_2^P}$-**c**)	Th. 11 / (12)	$\mathbf{\Sigma_4^P}$	Th. 9

Table 2: (Checking for existence of abstractions): Results on the computational complexity of abstracting by clustering. Decide whether there is a mapping m such that P is m-abstractable, given program P.

2 Preliminaries

We assume familiarity with basics in Boolean satisfiability (SAT) and satisfiability of quantified Boolean formulas (QSAT) [Kleine Büning and Lettman, 1999], graphs and digraphs [Diestel, 2024; Bang-Jensen and Gutin, 2008], computational complexity theory [Papadimitriou, 1994; Arora and Barak, 2009], and answer set programming [Janhunen and Niemelä, 2016; Calimeri *et al.*, 2020; Gebser *et al.*, 2012]. Below, we briefly recap notions used in this paper.

Computational Complexity. Let Σ and Σ' be finite alphabets. We call $I \in \Sigma^*$ an *instance* and n denotes the size of I. A *decision problem* is a subset $L \subseteq \Sigma^*$. Recall that P and NP are the complexity classes of all deterministically and non-deterministically polynomial-time solvable decision problems [Cook, 1971], respectively. We also need the Polynomial Hierarchy (PH) [Stockmeyer, 1976; Wrathall, 1976]. In particular, $\Delta_0^P := \Pi_0^P := \Sigma_0^P :=$ P and $\Delta_i^P := P^{\Sigma_{i-1}^P}$, $\Sigma_i^P := \text{NP}^{\Sigma_i^P}$, and $\Pi_i^P := \text{coNP}^{\Sigma_i^P}$ for $i > 0$ where C^D is the class C of decision problems augmented by an oracle for some complete problem in class D.

Answer Set Programming (ASP). We consider propositional programs. Let l, m, n be non-negative integers such that $l \leq m \leq n$, and $a_1, \ldots, a_n$ distinct propositional atoms. A *disjunctive rule* r is of the form

$$a_1 \vee \cdots \vee a_l \leftarrow a_{l+1}, \ldots, a_m, {\sim}a_{m+1}, \ldots, {\sim}a_n.$$

By $H_r \coloneqq \{a_1, \ldots, a_l\}$, $B_r^+ \coloneqq \{a_{l+1}, \ldots, a_m\}$, and $B_r^- \coloneqq \{a_{m+1}, \ldots, a_n\}$, we denote the *head*, *positive* or *negative body* atoms of r, respectively. We say that r is *normal* if $|H_r| \leq 1$, *positive* if $B_r^- = \emptyset$, and *Horn* if r is normal and positive. A *disjunctive logic program (DLP)* P, or *program* for short, is a set of rules, where $\mathrm{at}(P) \coloneqq \bigcup_{r \in P}(H_r \cup B_r^+ \cup B_r^-)$ denotes its atoms. For a set $\mathcal{U}$ of atoms and a logic program P, we say P is *over* $\mathcal{U}$ if $\mathrm{at}(P) \subseteq \mathcal{U}$. A program P has a certain property if all its rules have the property. The *(labeled) dependency digraph* $\mathcal{D}_P$ is the digraph defined on the set $\bigcup_{r \in P}(H_r \cup B_r^+ \cup B_r^-)$ of atoms, where for every rule $r \in P$, two atoms $b \in B_r^+ \cup B_r^-$ and $a \in H_r$ are joined by an edge (b, a) labeled by "$-$" if $b \in B_r^-$ and "$+$" if $b \in B_r^+$. An *odd cycle* of P is a cycle in $\mathcal{D}_P$ that contains odd number of negative labeled edges. A normal program is *stratified* if there is a mapping $\mathrm{str}\colon \mathrm{at}(P) \to \mathbb{N}$ such that for each rule $r \in P$, we have (i) if $x \in B_r^+$ and $y \in H_r$, then $\mathrm{str}(x) \leq str(y)$ and (ii) if $x \in B_r^-$ and $y \in H_r$, then $str(x) < str(y)$. By Disj, Strat, and Horn we denote the class of all disjunctive logic programs, stratified programs, and Horn programs respectively. An interpretation $M \subseteq \mathrm{at}(P)$ *satisfies* a rule r if $(H_r \cup B_r^-) \cap M \neq \emptyset$ or $B_r^+ \not\subseteq M$, and $M \models P$ if M satisfies every rule $r \in P$. The *(GL) reduct* of P with respect to M is defined as $P^M \coloneqq \{\, H_r \leftarrow B_r^+ \mid B_r^- \cap M = \emptyset \,\}$. Then, M is an *answer set* of P if M is a model of P, in symbols $M \models P$, such that no interpretation $N \subsetneq M$ is a model of P^M [Gelfond and Lifschitz, 1988]. We let the set $AS(P)$ consist of all answer sets of P. Deciding whether a program P has an answer set is Σ_2^P-c and in P for DLPs and stratified programs, respectively [Eiter and Gottlob, 1995; Gelfond and Lifschitz, 1988].

Abstractions

We follow standard notations on abstractions in ASP [Saribatur and Eiter, 2021; Saribatur *et al.*, 2024]. We admit to conveniently apply mappings also to sets of sets of atoms and rules. Below, we summarize existing notions of ASP abstractions. We denote the domain $\mathcal{D}$ of a function $f : \mathcal{D} \to \mathcal{A}$ by $\mathrm{dom}(f)$. For a function f, we denote the inverse function $f^{-1} \coloneqq \{\, f(d) \mapsto d \mid d \in \mathrm{dom}(f) \,\}$. Given sets $\mathcal{U}$ and $\mathcal{V}$ with $|\mathcal{U}| \geq |\mathcal{V}|$, a *mapping* m is a surjective function $m : \mathcal{U} \to \mathcal{V}$. For a set S, we let $m(S) \coloneqq \{\, m(e) \mid e \in S \,\}$. Next, we formally define what we mean by clustering atoms.

Definition 1. *We call a mapping $m : \mathcal{U} \to \mathcal{V}$ a* clustering *when $\mathcal{V}$ contains a set $\mathcal{V}_c = \{a \in \mathcal{V} \mid |m^{-1}(a)| > 1\}$ of* cluster atoms. *Accordingly, we divide $\mathcal{U}$ into* set of clustered atoms $Cl^m = \{a \in \mathcal{U} \mid m(a) \in \mathcal{V}_c\}$.

Next, we illustrate a clustering mapping.

Example 2. *Consider the following program*

$$P_0 \coloneqq \{d \vee e \leftarrow c. \quad f \leftarrow d. \quad f \leftarrow e\}.$$

Here, d and e both derive f whenever c holds true. So one can argue that specific details of whether d or e holds true are not of importance, and define a clustering as $m : \{d, e\} \to k$.[1]

[1]This notation of a mapping represents only the clustered atoms. k is a fresh (clustered) atom to which m maps multiple atoms to. For those atoms that are not clustered, i.e., remain singletons, we assume they are mapped to themselves.

Note that omission mapping [Saribatur and Eiter, 2021] can be seen as a special case of clustering, where the set of atoms to omit are clustered into $\top$.

Faithful abstractions. Faithful abstractions preserve exactly the projected semantics of the original program, i.e., they do not invent new answer sets that appear in the abstract program but have no counterpart in the original answer sets [Saribatur and Eiter, 2021].

Definition 3. *Formally, given a program P over $\mathcal{U}$ and a mapping $m : \mathcal{U} \to \mathcal{V}$, and a program Q over $\mathcal{V}$. Then, Q is a* faithful m-abstraction of P *if we have*

$$m(AS(P)) = AS(Q). \tag{1}$$

We say that P is faithfully m-abstractable *if there exists a Q that is a faithful m-abstraction of P.*

An ELP[2] P can always have a faithful abstraction, for any given mapping [Saribatur and Woltran, 2024]. Though faithful abstractions are too relaxed to capture an abstraction notion that preserves dependencies within the program. This was investigated with clustering mapping for non-ground programs, and the complexity of checking whether some program is a faithful abstraction of a given program was shown to be coNEXP^{NP}-complete [Saribatur *et al.*, 2021]. For ground programs, for omission mapping, checking faithfulness was shown to be Π_2^P-complete for normal programs [Saribatur and Eiter, 2021].

Example 4 (cont.)**.** *Consider program P_0 from Example 2. Naturally, since P_0 only has the empty set as the answer set, we can have $Q_0 = \emptyset$ as a faithful abstraction. If, however, we consider $P_0' \coloneqq P_0 \cup \{c\}$ that results in answer sets $\{c, d, f\}, \{c, e, f\}$, then program*

$$Q_0' = \{c. \quad k \leftarrow c. \quad f \leftarrow k\}$$

is a faithful abstraction of P_0', since $m(AS(P_0')) = \{\{c, k, f\}\} = AS(Q_0')$.

Uniform Abstractions. Uniform abstractions simplify a program by replacing atoms with abstract representatives uniformly across the entire program such that answer sets do not change even if arbitrary facts are added [Saribatur *et al.*, 2024].

Definition 5 (Uniform Abstractions)**.** *Formally, let $\mathcal{U}$ and $\mathcal{V}$ be disjoint sets of atoms such that $|\mathcal{V}| < |\mathcal{U}|$. Given a program P over $\mathcal{U}$ and a mapping $m : \mathcal{U} \to \mathcal{V}$, and a program Q over $\mathcal{V}$. Then, Q is a* uniform m-abstraction of P *if, for any set F of facts over $\mathcal{U}$, we have*

$$m(AS(P \cup F)) = AS(Q \cup m(F)). \tag{2}$$

We say that P is uniformly m-abstractable *if there exists a Q that is a uniform m-abstraction of P.*

Since not every program is uniform abstractable for a given mapping, Saribatur *et al.* (2024) studied necessary and sufficient conditions for abstractability in terms of SE- and UE-

[2]An extended logic program (ELP) P extends DLPs with double negation in the body.

models, and a model-based representation of such abstractions, showing that it is unique.[3]

Example 6 (cont.)**.** *Consider program P_0 from Example 2. We have $Q_0 = \{k \leftarrow c; f \leftarrow k\}$ as a uniform m-abstraction of P_0.*

Saribatur *et al.* (2024) established the following complexity results. Given a program P over $\mathcal{U}$ and a mapping $m : \mathcal{U} \to \mathcal{V}$. Deciding whether P is uniform m-abstractable is Π_3^P-complete. Given sets $\mathcal{U}$ and $\mathcal{V}$ of atoms, a program P over $\mathcal{U}$, a mapping $m : \mathcal{U} \to \mathcal{V}$, and a program Q over $\mathcal{V}$, checking whether Q is a uniform m-abstraction of P is (i) in Π_3^P and Π_2^P-hard if $P, Q \in$ Disj and (ii) the complexity drops to coNP when both programs are Horn.

3 Existence of Abstractions

Based on the definition of uniform abstractions, we first consider the computational problem for testing mappings, providing a more fine-grained analysis of previous considerations.

We start by assuming that the mapping m and program Q are given, and that we are unaware whether P is uniform m-abstractable.

Lemma 7. *Given a program P over $\mathcal{U}$, mapping $m : \mathcal{U} \to \mathcal{V}$, and a program Q. Deciding whether Q is a faithful m-abstraction of P is:*

1. *for stratified programs in P,*
2. *for normal programs Π_2^P-complete, and*
3. *for disjunctive programs Π_3^P-complete.*

Proof. ("Membership").
(Claim 1): Stratified programs can only have at most one answer set, which can be computed in polynomial time [Gelfond and Lifschitz, 1988].
(Claims 2 and 3): We show membership of the inverse problem. For membership of the inverse question, it suffices to guess a candidate M that is an answer set of Q such that all answer set candidates N with $N = m(M)$ that are not in $m(AS(P))$. Additionally, we might need to guess two answer set candidates M,N such that $N = m(M)$ is in $m(AS(P))$ while M is not an answer set of Q. Both checks work in $NP^{NP^{\mathcal{C}}}$, where $\mathcal{C}$ is the complexity for verifying a given answer set candidate. Consequently, the desired inverse problem is in the co-complexity class.
("Hardness").
(Claim 2): We reduce from $\forall\exists$-SAT, and let $\Phi = \forall U \exists V \varphi$ with $\varphi = \bigwedge_{i=1}^{n}(l_{i,1} \vee l_{i,2} \vee l_{i,3})$. We use copies of atoms, e.g., $\widetilde{U}=\{\widetilde{u} \mid u \in U\}$ and construct P_Φ as:

$$\begin{aligned} P_\Phi = \{&u \leftarrow not\ \widetilde{u}. \quad \widetilde{u} \leftarrow not\ u. \mid u \in U \cup V\} \cup \\ &\{usat \leftarrow \widetilde{l_{i,1}}, \widetilde{l_{i,2}}, \widetilde{l_{i,3}}. \mid c_i \in \varphi\} \cup \\ &\{\bot \leftarrow usat.\} \cup \{sat.\} \end{aligned}$$

where $\widetilde{l}_{i,j}$ is given by atom a if $l_{i,j}$ is $\sim a$ and by $\widetilde{a}$ if $l_{i,j}$ is a (positive) variable a. Q is constructed by:

$$Q = \{sat.\} \cup \{u \vee \widetilde{u}. \mid u \in U\}$$

We define mapping m by $m(u) = u$ for every $u \in U \cup \widetilde{U}$ and $m(v) = sat$ for $v \in \{sat\} \cup V \cup \widetilde{V} \cup W \cup \widetilde{W}$. It is easy to see that Φ is valid iff Q is a faithful m-abstraction of P_Φ.
(Claim 3): We reduce from $\forall\exists\forall$-SAT, and let $\Phi = \forall U \exists V \forall W \varphi$ with $\varphi = \bigvee_{i=1}^{n}(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$. We construct P_Φ as follows.

$$\begin{aligned} P_\Phi = \{&u \vee \widetilde{u}. \mid u \in U \cup V \cup W\} \cup \\ &\{sat \leftarrow \widetilde{l_{i,1}}, \widetilde{l_{i,2}}, \widetilde{l_{i,3}}. \mid d_i \in \varphi\} \cup \\ &\{w \leftarrow sat., \widetilde{w} \leftarrow sat. \mid w \in W\} \cup \{\bot \leftarrow \sim sat.\} \end{aligned}$$

where $\widetilde{l}_{i,j}$ is given by atom $\widetilde{a}$ if $l_{i,j}$ is $\sim a$ and by a if $l_{i,j}$ is a (positive) variable a. Q is constructed by:

$$Q = \{sat.\} \cup \{u \vee \widetilde{u}. \mid u \in U\}$$

We define mapping m by $m(u) = u$ for every $u \in U \cup \widetilde{U}$ and $m(v) = sat$ for $v \in \{sat\} \cup V \cup \widetilde{V} \cup W \cup \widetilde{W}$. Then, Φ is valid iff Q is a faithful m-abstraction of P_Φ. □

Remark that there is a complexity jump from stratified programs to normal programs. This is because, for membership, it does not suffice to guess a set $S \subseteq U$ that is not an answer set of P but $m(S)$ is an answer set of Q, as there can be other $S' \subseteq U$ that is an answer set of P and $m(S) = m(S')$. In fact, for every S' that can be mapped to $m(S)$, one needs to check that S' is not an answer set of P. This increases complexity.

This leads to the following result.

Theorem 8. *Given a program P over $\mathcal{U}$, mapping $m : \mathcal{U} \to \mathcal{V}$, and a program Q. Deciding whether Q is a uniform m-abstraction of P is:*

1. *for stratified programs coNP-complete,*
2. *for normal programs in Π_2^P, and*
3. *for disjunctive programs Π_3^P-complete.*

Proof. ("Membership"):
The upper bound was already established by Saribatur and Woltran (2024) for disjunctive programs. The construction works as follows: For given P, Q, and m, we check for Equation (2) defining Q to be a uniform m-abstraction. For the complementary problem, we guess a set F of facts and a set I checking whether $I \in AS(P \cup F)$ and checking whether $m(I) \in AS(Q \cup m(F))$, but that it does not hold for both. As answer set computation for disjunctive programs is in Π_2^P, the equality check is contained in Σ_3^P. This then drops down one level for normal programs, and two levels for stratified programs.
("Hardness"):
(Claim 1): We reduce from $\forall$-SAT, and let $\Phi = \forall U \varphi$ with $\varphi = \bigvee_{i=1}^{n}(l_{i,1} \wedge l_{i,2} \wedge l_{i,3})$.

$$\begin{aligned} P_\Phi = \{&\overline{d}_i \leftarrow \widetilde{l_{i,j}}. \mid d_i \in \varphi\} \cup \\ &\{usat \leftarrow \overline{d}_i, l_{i,1}, l_{i,2}, l_{i,3}. \mid d_i \in \varphi\} \cup \end{aligned}$$

[3] SE-models, also referred as HT-models [Lifschitz *et al.*, 2001], represent the monotone core of the program and UE-models capture their maximal proper subsets [Eiter *et al.*, 2007]. They are used to characterize strong and uniform equivalence of programs.

$$\{\leftarrow \sim usat, \overline{d}_1, \ldots, \overline{d}_{|\varphi|}\}$$

where $\widetilde{l}_{i,j}$ is given by atom a if $l_{i,j}$ is $\sim a$ and by $\sim a$ if $l_{i,j}$ is a (positive) variable a.

Q is independent from Φ and constructed by:

$$Q = \{sat.\}$$

We define mapping m by $m(u) = sat$ for every $u \in U \cup \{usat, \overline{d}_i \mid 1 \leq i \leq |\Phi|\}$. It is easy to see that Φ is valid iff Q is a uniform m abstraction of P_Φ.

(Claim 3): We establish the lower bound by knowing that uniform simplifiability [Saribatur and Woltran, 2023], which abstraction by omission of atoms, is a special case of clustering see [Saribatur *et al.*, 2024, Prop.16]. In [Saribatur and Woltran, 2023] Π_3^P-hardness proof of checking whether P is simplifiable for a given set A of atoms, a program P_Φ is constructed where A cannot be omitted if and only if the $(3, \exists)$-QSAT formula Φ is satisfied. In [Saribatur and Woltran, 2023] it was also shown that if P is uniform abstractable for omitting A, then $P_{|\overline{A}}$ is an abstraction where $\overline{A} = \mathcal{U} \setminus A$ and $P_{|\overline{A}}$ refers to restricting the vocabulary to $\overline{A}$. Thus, for hardness, we can use that P_Φ and construct Q as $P_{|\overline{A}}$ which is then not an uniform abstraction for omitting A of P if and only if Φ is true. □

Ultimately, we are interested in whether a mapping m even exists so that P is uniform m-abstractable, as it is known that not every program might be abstractable with a *non-trivial mapping*, meaning that it maps at least two atoms to one.

Theorem 9. *Given a program P over $\mathcal{U}$. Deciding whether there exists a non-trivial mapping $m : \mathcal{U} \to \mathcal{V}$ such that P is uniform m-abstractable.*

1. *for stratified programs in Σ_2^P,*
2. *for normal programs in Σ_3^P, and*
3. *for disjunctive programs in Σ_4^P.*

Proof. It is easy to see that we can guess m with an additional level (on top of Theorem 8). Indeed, by [Saribatur *et al.*, 2024] we know that the semantic characterization of uniform abstractions is unique, thus we can restrict ourselves to specific programs Q that are determined by P and m. □

While Saribatur and Woltran (2024) observed that for any set A of atoms, an ELP P is faithful abstractable by omitting A, this does not hold for programs (instead of ELPs) and clustering. Below, Example 10 illustrates that if we consider programs, it is not always the case that a faithful abstraction by clustering exists.

Example 10. *Consider the following program.*

$$P_1 := \{a \vee c. \quad b \leftarrow a.\}.$$

The answer sets of P_1 are $\{a, b\}$ and $\{c\}$. Now, consider the mapping $m : \{a, c\} \mapsto k$. Then, the desired answer set of a faithful abstraction would be $\{k, b\}$ and $\{b\}$, which however conflicts with subset-minimality of answer sets. In turn, with the mapping m a faithful abstraction cannot be found.

In consequence of the previous example, we can now ask how hard it is to find a mapping such that P is faithful m-abstractable.

It turns out that the decision problem just boils down to answer set consistency, however, we prove even a stronger result. Even if we fix $\mathcal{V}$, a faithful abstraction trivially exists.

Theorem 11. *Given a program P over $\mathcal{U}$ and a set $\mathcal{V}$ of atoms. There always exists a non-trivial mapping $m : \mathcal{U} \to \mathcal{V}$ s.t. P is faithful m-abstractable (decidable in constant time).*

Proof. We show that a witnessing program Q and matching m always exists. If the empty answer set is an answer set of P, observe that this has to be the only answer set, as disjunctive rules do not allow both an empty answer set and a non-empty answer set simultaneously. Consequently, in this case we construct $Q = \{\leftarrow x, \neg x. \mid x \in \mathcal{V}\}$ and take any mapping m.

Otherwise, if P does not have an answer set, we let $Q = \{\leftarrow \neg x. \mid x \in \mathcal{V}\}$ and take any arbitrary mapping m.

Otherwise, if P admits a (non-empty) answer set, we pick an arbitrary $y \in \mathcal{V}$ and construct $Q = \{y.\} \cup \{\leftarrow x, \neg x. \mid x \in \mathcal{V}, x \neq y\}$ and set $m(a) = y$ for every $a \in \mathcal{U}$. □

However, if we ask for a consistent Q, the situation changes.

Theorem 12. *Given a program P over $\mathcal{U}$ and a set $\mathcal{V}$ of atoms. Asking whether there is exists a non-trivial mapping $m : \mathcal{U} \to \mathcal{V}$ s.t. P is faithful m-abstractable for some consistent Q is:*

1. *for stratified programs in P-complete,*
2. *for normal programs in* NP*-complete, and*
3. *for disjunctive programs in Σ_2^P-complete.*

Proof. ("Membership").
If the empty answer set is an answer set of P, which is in Σ_i^P for stratified ($i = 0$), normal ($i = 0$), and disjunctive ($i = 1$) programs, as above we construct witness $Q = \{\leftarrow x. \mid x \in \mathcal{V}\}$ and take any mapping m.

Otherwise, we return yes iff P admits a (non-empty) answer set, as this allows us to pick an arbitrary $y \in \mathcal{V}$ and construct witness $Q = \{y.\} \cup \{\leftarrow x. \mid x \in \mathcal{V}, x \neq y\}$ and set $m(a) = y$ for every $a \in \mathcal{U}$.

("Hardness").
We pick any $a \in \mathcal{U}$ and set $\mathcal{V} = \{a\}$. Suppose that there is a Q and an $m : \mathcal{U} \to \mathcal{V}$ such that $m(AS(P)) = AS(Q)$. Since Q admits an answer set by assumption, we know that our abstraction problem decides the consistency of P, as then P is consistent as well. □

Constructing Abstractions

Now after we established knowledge about the computational complexity of checking for existence of abstractions, we ask how to come up with the concrete abstraction other than the very generic consideration as in the previous section. Interestingly, Saribatur *et al.* (2024) introduced a syntactic operator that simply applies the mapping m to the atoms in each rule in P, denoted as $m(P)$, to construct uniform abstractions as in the following example.

Example 13 (cont.)**.** *Consider program P_0 from Example 2. The program $m(P_0)$ is $\{k \leftarrow c; f \leftarrow k\}$, which is a uniform m-abstraction.*

This syntactic operator is guaranteed to work for positive programs or for programs where negation can be eliminated, while the guarantee no longer holds for more general ones.

Next, we address this topic by introducing a new operator that allows us to construct uniform abstractions of disjunctive programs. We observe that the construction $m(P)$ fails to achieve abstractions if the clustered atoms have negative dependencies, though without odd loops.

Example 14. *Consider the programs*

$$P_2 \coloneqq \{a \leftarrow \sim b.\} \text{ and } \quad P_2' \coloneqq \{a \leftarrow \sim b. \quad b \leftarrow \sim a.\}$$

When clustering a and b by $m{:}\{a,b\} \mapsto k$, $m(P_2) = m(P_2') = \{k \leftarrow \sim k\}$ is clearly not a uniform m-abstraction of the programs, since the negative dependency among the atoms is overwritten. However, it can be observed that $Q = \{k.\}$ is a uniform m-abstraction of both programs. Although, for the program $P_2'' = P_2 \cup \{b \leftarrow a\}$, the program $\{k \leftarrow \sim k\}$ would be a uniform m-abstraction.

Next, we extend the previous operator to treat negative dependency. For a program P with exactly l odd loops, let $L_{c_1}, \ldots, L_{c_l}$ denote all sets of literals of an odd loop, if exists.

Definition 15. *Given a mapping m and a rule $r : H_r \leftarrow B_r$,*

$$m^\star(r) \coloneqq m(H_r) \leftarrow m(B_r^+), m(B_r^- \setminus L)$$

where $L \subseteq B_r^- \cap Cl^m$ satisfying $\forall \ell \in L\ \exists \alpha \in H_r : m(\alpha) = m(\ell)$ and $\{\alpha, \ell\} \not\subseteq L_{c_j}$ for all j.

By applying $m^\star$ to P*, we mean $m^\star(P) \coloneqq \bigcup_{r \in P} m^\star(r)$.*

The intention of the operator is to keep track of the odd loops, and remove those atoms from the negative body that agree on the mapping with the head of the rule but are not involved in an odd loop.

Example 16 (cont.)**.** *Consider program P_2 from Example 14. Applying m^* to rule $r_1 = \{a \leftarrow \sim b\}$ gives us $m^*(r_1) = \{m(a) \leftarrow\}$ since $\{b\} \subseteq B_{r_1}^- \cap Cl^m$, $a \in H_{r_1}$, $m(a) = m(b)$. Similarly for $r_2 = \{b \leftarrow not\ a\}$ from P_2'. Thus we get $m^\star(P_2) = m^\star(P_2') = \{k\}$. Now for program $P_2'' = P_2 \cup \{b \leftarrow a\}$ we have $L_c = \{b, a\}$ as the set of literals involved in an odd loop. Thus $m^\star(P_2'') = \{k \leftarrow \sim k; k \leftarrow k\}$.*

Saribatur *et al.* (2024) established that not every program is uniform abstractable for a given mapping. To find the necessary and sufficient conditions for abstractability on the SE-models of the program, they use the following observation.

Proposition 17 (Saribatur *et al.* 2024)**.** *If there is a uniform m-abstraction of P, then, for all Y and all sets of facts Z, Z' such that $m(Z') = m(Z)$:*

$$\text{If } Y \in AS(P \cup Z), \text{ then} \qquad (3)$$
$$\exists Y' \text{ s.t. } m(Y') = m(Y) \text{ and } Y' \in AS(P \cup Z')$$

Instead of shifting the focus to SE-models, we remain with the answer set correspondence checking as it provides a much simpler picture. Thus, we first show that this condition is sufficient for the existence of a program Q, by showing that $Q = m^\star(P)$ is a uniform m-abstraction and thus satisfies Equation (2) of Definition 5.[4]

[4]We prove statements marked by "$\star$" in the supplemental material at https://www.dbai.tuwien.ac.at/user/saribat/pub/ijcai26-supp.pdf.

Theorem 18 ($\star$)**.** *Let $\mathcal{U}$ and $\mathcal{V}$ be sets of atoms, P be a program over $\mathcal{U}$ and $m : \mathcal{U} \to \mathcal{V}$ a mapping. If P satisfies Equation* (3) *for mapping m, then $Q = m^\star(P)$ is a uniform m-abstraction of program P.*

Proof (Sketch). We assume this is not the case, and that there exists some F such that $m(AS(P \cup F)) \neq AS(Q \cup m(F))$. We do case analysis and first look at the case when there exists $I' \in AS(P \cup F)$, while $m(I') = I \notin AS(Q \cup m(F))$. Since the aim is to reach a contradiction to existence of an $I' \in AS(P \cup F)$ with $m(I') = I$ it is enough to look at $F' = F \cup m^{-1}(m(F))$ which contains all atoms clustered in $m(F)$. Then since P satisfies (3), we reach that this condition holds for all F. By looking at the possibilities of I not being an answer set of $AS(Q \cup m(F))$ by tracing the trouble making rules, we always reach a contradiction. The same approach is applied for the other case when There exists $I \in AS(Q \cup m(F))$ s.t. $I \notin m(AS(P \cup F))$. □

This operator thus extends our ability to construct uniform abstractions to programs, whenever possible.

Observe that, Theorem 18 shows that if Equation (3) is satisfied for a mapping, then a uniform abstraction can be constructed. Thus Equation (3) is in fact a *necessary and a sufficient condition* for the existence of a uniform abstraction.

Corollary 19. *Given a program P and a mapping m, P satisfies Equation* (3) *for mapping m iff there is a uniform m-abstraction of P.*

4 Interchangeable vs. Indistinguishable

Is it possible to abstract over symmetries? That is a natural question that arises from abstractions and when constructing operators. In this section, we take a look at symmetries and investigate their potential to obtain an abstracted program in the context of uniform and faithful abstractions.

4.1 Symmetry

We begin by providing the usual definitions of symmetry from the context of propositional satisfiability [Krishnamurthy, 1985] to ASP. A *permutation* of a set S is a bijection $\sigma : S \to S$, meaning that σ rearranges the elements of S in which each element i is replaced by the corresponding $\sigma(i)$. For example, $(3, 1, 2)$ corresponds to $\sigma(1) = 3$, $\sigma(2) = 1$, and $\sigma(3) = 2$.

Definition 20 (Symmetry)**.** *Given a program P over $\mathcal{U}$ and a permutation $\sigma : \mathcal{U} \to \mathcal{U}$. The permutation σ is*

- *a* syntactic symmetry *of program P if P and $\sigma(P)$ are identical;*
- *a* semantic symmetry *of P if $AS(P) = AS(\sigma(P))$.*

It is known that every syntactic symmetry is a semantic symmetry, and not vice versa, which we also illustrate in the following example.

Example 21 (cont.)**.** *Consider program P_1 from Example 10. Consider permutation $\sigma = (b, a)$ meaning $a \mapsto b, b \mapsto a$. The program $\sigma(P_1)$ is not identical to P_1 and thus σ is not a syntactical symmetry. But σ is a semantic symmetry, since a and b always appear together in an answer set.*

As we are interested in investigating abstraction over symmetry, we define the notions of symmetry between atoms in a program. We use cycles where a permutation is a product of disjoint cycles. A cycle $a_1 a_2 a_3 \dots a_k$ means that the permutation maps $a_1 \mapsto a_2$, $a_2 \mapsto a_3$, and so on, finally $a_k \mapsto a_1$.

Definition 22 (Symmetry of Atoms)**.** *Given a program P over $\mathcal{U}$. Then, a set $S \subseteq \mathcal{U}$ of atoms is*

- syntactically symmetric *to one another in P, if there exists a permutation σ with cycle $\ell_1 \dots \ell_n$ for $\ell_i \in S$ that is a syntactic symmetry of P.*
- semantically symmetric *to one another in P, if there exists a permutation σ with cycle $\ell_1 \dots \ell_n$ for $\ell_i \in S$ that is a semantic symmetry of P.*

The following example illustrates symmetry of atoms.

Example 23. *Consider the following programs.*

$$P_3 \coloneqq \{a \vee b. \quad c \leftarrow a. \quad c \leftarrow b.\}$$
$$P_4 \coloneqq P_3 \cup \{b \leftarrow a.\}$$

For the set $S = \{a, b\}$, we can see that the atoms in S are syntactically and thus semantically symmetric since the permutation $\sigma = (a, b)$ is a syntactic symmetry of P_3. However for program P_4, the atoms in S are neither syntactically nor semantically symmetric.

The next example illustrates that actually not every semantically symmetric is a syntactically symmetric.

Example 24. *In program $P_5 = \{b \leftarrow a\}$, a and b are semantically symmetric, as P_5 has only empty set as answer set, while not syntactically symmetric.*

We have seen that uniform abstractions are able to abstract over atoms that are semantically indistinguishable according to their dependencies within the program. One natural question that arises is whether symmetric atoms have such an indistinguishable aspect, which seem to be not always the case as we illustrate in the example

Example 25 (cont.)**.** *Consider programs P_3 and P_4 from Example 23. For a mapping $m : \{a, b\} \to k$, the program*

$$Q = \{k. \quad c \leftarrow k\}$$

is a uniform abstraction for both of them.

The above example shows that uniform abstractions are more general than capturing indistinguishability w.r.t syntactic symmetry, since in case of P_4 the existence of $\{b \leftarrow a.\}$ causes to lose the syntactic symmetry property, while still presents an indistinguishable behavior for abstracting.

Though also not every symmetry is uniform abstractable.

Example 26. *Consider the program as follows.*

$$P_6 \coloneqq \{a \vee b. \quad c \leftarrow {\sim}b. \quad c \leftarrow {\sim}a\}$$

a and b are syntactically symmetric, and thus semantically symmetric, however not uniform abstractable as we have $AS(P_6 \cup \{a, b\}) = \{\{a, b\}\}$ while $AS(P_6 \cup \{a\}) = \{\{a, c\}\}$, not agreeing on the mapping $\{a, b\} \to k$.

Consider the program as follows.

$$P_7 \coloneqq \{a \vee b. \quad c \leftarrow a, b.\}$$

Similar to P_6, here, a and b are syntactically symmetric, and thus semantically symmetric. However, they are not uniform abstractable, since we have $AS(P_7 \cup \{a, b\}) = \{a, b, c\}$ while $AS(P_7 \cup \{a\}) = \{a\}$, not agreeing on the mapping $\{a, b\} \mapsto k$.

In the above example, although a and b are symmetric, they are not semantically indistinguishable, since c depends on their individual truth values, thus an abstraction satisfying the uniform condition cannot be achieved. However, for both programs P_6 and P_7 it is possible to come up with a faithful abstraction, as they are less restrictive than uniform abstractions.

Example 27 (cont.)**.** *Consider programs P_6 and P_7 from Example 26. The programs $Q_6 = \{k. \quad c \leftarrow k\}$ and $Q_7 = \{k\}$ are faithful abstractions for programs P_6 and P_7, satisfying $m(AS(P_i)) = AS(Q_i), i \in \{6, 7\}$.*

However, in some cases, the obtained faithful abstraction might no longer resemble the original structure.

Example 28. *Consider the program P_8 as follows.*

$$P_8 \coloneqq \{a \vee b. \quad x \vee y. \quad a \leftarrow x. \quad b \leftarrow x. \quad c \leftarrow a, b\}.$$

We have $AS(P_8) = \{\{x, a, b, c\}, \{y, a\}, \{y, b\}\}$. a and b are syntactically symmetric, and the program

$$Q_8 = \{k. \quad x \vee y. \quad k \leftarrow x. \quad c \leftarrow k, {\sim}y\}.$$

is a faithful abstraction of P_8. Observe that the last rule in Q_8 needs to be newly constructed to capture the semantic of the last rule of P_8, thus no longer resembles it.

4.2 Indistinguishability

As shown, not every symmetry is uniform abstractable, meaning that uniform abstractions are too strong. Although faithful abstractions are weaker and could be obtained, it is not clear how to construct them when implicit relations that abstraction cannot distinguish need to be preserved in some way, as illustrated in Example 28. Meanwhile, our operator $m^\star$ in Definition 15 gives an intuitive way to construct abstractions with clustering. Thus, we can naturally ask whether it can be used to obtain intuitive faithful abstractions, which showed to be of importance when wanting to abstract over symmetry. Therefore, we first examine the following example.

Example 29 (cont.)**.** *Consider programs P_6 and P_7 from Example 26. The program $m^\star(P_6) \coloneqq \{k. \quad c \leftarrow {\sim}k.\}$, resp. $m^\star(P_7) \coloneqq \{k. \quad c \leftarrow k.\}$, is not a faithful abstraction of P_6, resp. P_7.*

In program P_7, a and b both need to hold true to obtain c, which actually never occurs, while in P_6 at most one of a or b needs to hold true to get c. These relations cannot be handled with such a syntactic operator. However, such cases can be detected when looking at the addition of a and b as facts, as shown in Example 26. Thus, we define a property over P, which is a relaxation of Equation (3) to catch such cases.

Definition 30. *Given program P and mapping $m : \mathcal{U} \to \mathcal{V}$ with $k \in \mathcal{V}_c$. We say that the set $S = m^{-1}(k)$ of atoms are* semantically indistinguishable *in P with m if for all Y and all sets of facts $Z, Z' \subseteq S$:*

$$\text{If } Y \in AS(P \cup Z), \text{ then} \qquad (4)$$
$$\exists Y' \text{ such that } m(Y') = m(Y) \text{ and } Y' \in AS(P \cup Z')$$

Example 31 (cont.)**.** *In Example 26, for the programs P_6 and P_7, the atoms a and b are not semantically indistinguishable, while in P_3 from Example 23 they are.*

It turns out we can use $m^\star$ to obtain faithful abstractions over the semantically indistinguishable.

Theorem 32 ($\star$)**.** *Given a program P over $\mathcal{U}$ and a mapping m, if all clustered atoms are semantically indistinguishable, then, the program $m^\star(P)$ is a faithful abstraction.*

4.3 From Symmetry to Abstraction

Let us now investigate how and which abstractions are possible over the symmetry. For this the notion of m-positive programs, that disallows occurrence of clustered atoms in negative body of the rules, comes handy.

Definition 33 (Saribatur *et al.* 2024)**.** *For a mapping m, program P is* m-positive *if for each rule $r \in P$, $B_r^- \cap Cl^m = \emptyset$.*

Our observations on programs P_6 and P_7 from Example 26 show, in addition to m-positiveness, also the need to disallow occurrence of clustered atoms in the same body. Thus we establish the following result. Here, for simplicity, we focus on clustering only two atoms.

Proposition 34 ($\star$)**.** *Let P be a program with syntactically symmetric atoms a_1, a_2. For a mapping m:$\{a_1, a_2\} \mapsto k$, if P is m-positive and if there is no rule $r \in P$ with $\{a, b\} \subseteq B_r$ then a_1 and a_2 are semantically indistinguishable in P.*

In fact, the syntactic conditions over P allows us to reach uniform abstractability from syntactic symmetry.

Theorem 35 ($\star$)**.** *Let P be a program with syntactically symmetric atoms a_1, a_2. For a mapping m:$\{a_1, a_2\} \mapsto k$, if P is m-positive and if there is no rule $r \in P$ with $\{a, b\} \subseteq B_r$ then P is uniform m-abstractable.*

Discussion

Intuitively, the notion of being semantically indistinguishable captures situations when we only need to match existence, thus we do not care about the actual object. Whereas interchangeable talks about whether there is a permutation that allows us to swap objects. The idea of *symmetry* is to detect interchangeable objects, thus avoiding an explosion in the search for a potential solution, such as pigeons and holes in the pigeonhole problem or colors in the graph coloring problem. However, in these problems, even though the objects are interchangeable, identifying them is necessary to ensure that the problem's constraints are still satisfied. This can also be observed in a very recent work [Carbonnelle *et al.*, 2024] where the authors intuitively compress these objects while still preserving the information about the number of objects.

Abstraction, on the other hand, is about losing the identification of the objects as they do not matter in finding a solution. For example, consider having checkpoints to pass through in a package delivery problem or having multiple tables to use when moving the blocks in the blocksworld problem. In the former problem, the choice of which middle point the truck passes through is not important and the same holds for the choice of tables in the latter problem. illustrates how for these objects in the above problems, symmetry breaking *cannot help*, even if one considers the best possible constraint that steers the solving to only one solution by forbidding choosing an action that uses a non-maximal object among the ordered symmetrical objects. Meanwhile abstracting over these objects helps. The reason behind is that even though one breaks the symmetries, n objects remain in the search space blowing up the search, while with abstraction the number of objects are reduced to one.

dom.	inst.	obj.	w/o Sym.Br	w/ Sym.Br.	w/ Abs.
bw$^+$	b3	500	5.6	2.8	0.1
		1000	23.2	11.8	0.0
		5000	519.2	310.2	0.0
	b7	500	204.3	31.5	0.1
		1000	3460.2	130.6	0.1
		5000	404.6	401.0	0.1
nm$^+$	p1	500	1.8	1.8	0.1
		1000	7.0	6.6	0.1
		5000	180.5	178.4	0.0
	p6	500	521.7	169.3	0.3
		1000	1689.2	507.6	0.2
		5000	mo	mo	0.2

Table 3: Comparison of breaking vs. abstracting the symmetry: nm$^+$ and bl$^+$ refer to the nomystery (package delivery) and blocksworld problems, with adding checkpoints and multiple tables, respectively. Column "inst." refers to the number of blocks (b) and packages (p) in these problems. Column "obj." refers to the number of indistinguishable objects, i.e., checkpoints and tables, in these problems. Solving (including grounding) times in seconds. Resource limits: 7200s runtime and 25GB main memory. *mo* means memory out.

5 Conclusion

In this paper, we investigate faithful and uniform abstractions, and their computational complexity. We provide concrete operators for constructing abstractions. Our investigations reveal that for cases of interchangeability, obtaining a uniform abstraction with generalization ability is not feasible; however, faithful abstractions can still be obtained, albeit they may not closely resemble the original program. The notion of semantic indistinguishability captures the cases where it is possible to use the intuitive operator to achieve (faithful) abstractions. Our results indicate that symmetry and abstraction are orthogonal. Symmetry aims to detect (and find ways to reconstruct) interchangeable objects. Abstraction aims at losing the identification of objects. However we show that it is possible to abstract over symmetrical atoms with additional syntactic properties which ensure indistinguishability, and show a case study on its usefulness compared to symmetry breaking which motivates further research into automating the search for indistinguishable symmetries.

Indeed, while we closed open questions in complexity, there are still open gaps; we expect that this work will foster follow-up studies on different versions of abstractions useful for generalization and their complexity.

Acknowledgments

This research was funded in whole or in part by the Austrian Science Fund (FWF) grant 10.55776/T1315, the Vienna Science and Technology Fund (WWTF) grant 10.47379/ICT25044 (Saribatur); the Austrian Science Fund (FWF) grant J4656, the French Agence nationale de la recherche (ANR) grant ANR-25-CE23-7647 (Hecher); and Excellence Center at Linköping – Lund in Information Technology (ELLIIT) funded by the Swedish government and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation (Fichte).

References

[Alviano *et al.*, 2015] Mario Alviano, Carmine Dodaro, Nicola Leone, and Francesco Ricca. Advances in WASP. In *LPNMR'15*, pages 40–54. Springer, 2015.

[Alviano *et al.*, 2017] Mario Alviano, Francesco Calimeri, Carmine Dodaro, Davide Fuscà, Nicola Leone, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. The ASP system DLV2. In *LPNMR'17*, pages 215–221. Springer, 2017.

[Arora and Barak, 2009] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[Bang-Jensen and Gutin, 2008] Jørgen Bang-Jensen and Gregory Z. Gutin. *Digraphs*. Springer, 2008.

[Banihashemi *et al.*, 2017] Bita Banihashemi, Giuseppe De Giacomo, and Yves Lespérance. Abstraction in situation calculus action theories. In *AAAI'17*, pages 1048–1055, 2017.

[Benhamou, 2013] Belaïd Benhamou. Dynamic and static symmetry breaking in answer set programming. In *LPAR'13*, pages 112–126, Berlin, Heidelberg, 2013.

[Berthold *et al.*, 2019] Matti Berthold, Ricardo Gonçalves, Matthias Knorr, and João Leite. A syntactic operator for forgetting that satisfies strong persistence. *Theory and Practice of Logic Programming*, 19(5-6):1038–1055, 2019.

[Berthold, 2022] Matti Berthold. On syntactic forgetting with strong persistence. In *KR'22*, pages 43–52. IJCAI Organization, 2022.

[Brass and Dix, 1997] Stefan Brass and Jürgen Dix. Characterizations of the disjunctive stable semantics by partial evaluation. *The Journal of Logic Programming*, 32(3):207–228, 1997.

[Brewka *et al.*, 2011] Gerhard Brewka, Thomas Eiter, and Mirosław Truszczyński. Answer set programming at a glance. *Comm. of the ACM*, 54(12):92–103, 2011.

[Calimeri *et al.*, 2020] Francesco Calimeri, Wolfgang Faber, Martin Gebser, Giovambattista Ianni, Roland Kaminski, Thomas Krennwallner, Nicola Leone, Marco Maratea, Francesco Ricca, and Torsten Schaub. ASP-Core-2 input language format. *Theory and Practice of Logic Programming*, 20(2):294–309, 2020.

[Carbonnelle *et al.*, 2024] Pierre Carbonnelle, Gottfried Schenner, Maurice Bruynooghe, Bart Bogaerts, and Marc Denecker. Using symmetries to lift satisfiability checking. In *AAAI'24*, pages 7961–7968. AAAI Press, 2024.

[Cook, 1971] Stephen A. Cook. The Complexity of Theorem-Proving Procedures. In *STOC'71*, pages 151–158. ACM, 1971.

[Cui *et al.*, 2021] Zhenhe Cui, Yongmei Liu, and Kailun Luo. A uniform abstraction framework for generalized planning. In *IJCAI'21*, pages 1837–1844, 2021.

[Devriendt and Bogaerts, 2016] Jo Devriendt and Bart Bogaerts. Breakid: Static symmetry breaking for ASP (system description). *CoRR*, abs/1608.08447, 2016.

[Devriendt *et al.*, 2016] Jo Devriendt, Bart Bogaerts, Maurice Bruynooghe, and Marc Denecker. Improved static symmetry breaking for SAT. In *SAT'16*, volume 9710 of *LNCS*, pages 104–122. Springer, 2016.

[Diestel, 2024] Reinhard Diestel. *Graph Theory*. Springer, 2024.

[Doherty and Szałas, 2024] Patrick Doherty and Andrzej Szałas. Dual forgetting operators in the context of weakest sufficient and strongest necessary conditions. *Artificial Intelligence*, 326:104036, 2024.

[Dong *et al.*, 2025] Hao Dong, Zheyuan Shi, Hemeng Zeng, and Yongmei Liu. An automatic sound and complete abstraction method for generalized planning with baggable types. In *AAAI'25*, volume 39, pages 14875–14884, 2025.

[Drescher *et al.*, 2011] Christian Drescher, Oana Tifrea, and Toby Walsh. Symmetry-breaking answer set solving. *AI Commun.*, 24(2):177–194, 2011.

[Eiter and Gottlob, 1995] Thomas Eiter and Georg Gottlob. On the computational cost of disjunctive logic programming: Propositional case. *Annals of Mathematics and Artificial Intelligence*, 15(3–4):289–323, 1995.

[Eiter *et al.*, 2004] Thomas Eiter, Michael Fink, Hans Tompits, and Stefan Woltran. Simplifying logic programs under uniform and strong equivalence. In *LPNMR'04*, pages 87–99. Springer, 2004.

[Eiter *et al.*, 2007] Thomas Eiter, Michael Fink, and Stefan Woltran. Semantical characterizations and complexity of equivalences in answer set programming. *ACM Transactions on Computational Logic (TOCL)*, 8(3):17, 2007.

[Fox *et al.*, 2005] Maria Fox, Derek Long, and Julie Porteous. Abstraction-based action ordering in planning. In *IJCAI'05*, 2005.

[Gebser *et al.*, 2012] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Answer set solving in practice. *Synthesis lectures on artif. intell. and machine learning*, 6(3):1–238, 2012.

[Gebser *et al.*, 2014] Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Clingo = ASP + control: Preliminary report. *CoRR*, abs/1405.3694, 2014.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In *ICLP/SLP'88*, volume 2, pages 1070–1080. MIT Press, August 1988.

[Giacomo *et al.*, 2025] Giuseppe De Giacomo, Yves Lespérance, and Matteo Mancanelli. Situation calculus temporally lifted abstractions for generalized planning. In *AAAI'25*, pages 14848–14857. AAAI Press, 2025.

[Gonçalves *et al.*, 2021] Ricardo Gonçalves, Tomi Janhunen, Matthias Knorr, and João Leite. On syntactic forgetting under uniform equivalence. In Wolfgang Faber, Gerhard Friedrich, Martin Gebser, and Michael Morak, editors, *JELIA'21*, volume 12678 of *LNCS*, pages 297–312. Springer, 2021.

[Gonçalves *et al.*, 2023] Ricardo Gonçalves, Matthias Knorr, and João Leite. Forgetting in answer set programming - A survey. *Theory Pract. Log. Program.*, 23(1):111–156, 2023.

[Illanes and McIlraith, 2019] León Illanes and Sheila A McIlraith. Generalized planning via abstraction: Arbitrary numbers of objects. In *AAAI'19*, volume 33, pages 7610–7618, 2019.

[Janhunen and Niemelä, 2016] Tomi Janhunen and Ilkka Niemelä. The answer set programming paradigm. *AI Magazine*, 37(3):13–24, 2016.

[Janhunen *et al.*, 2006] Tomi Janhunen, Ilkka Niemelä, Dietmar Seipel, Patrik Simons, and Jia-Huai You. Unfolding partiality and disjunctions in stable model semantics. *ACM Transactions on Computational Logic (TOCL)*, 7(1):1–37, 2006.

[Kleine Büning and Lettman, 1999] Hans Kleine Büning and Theodor Lettman. *Propositional Logic: Deduction and Algorithms*, volume 48 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, 1999.

[Krishnamurthy, 1985] Balakrishnan Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 22(3):253–275, 1985.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustín Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic (TOCL)*, 2(4):526–541, October 2001.

[Nebel *et al.*, 1997] Bernhard Nebel, Yannis Dimopoulos, and Jana Koehler. Ignoring irrelevant facts and operators in plan generation. In *ECP'97*, pages 338–350. Springer, 1997.

[Papadimitriou, 1994] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[Pearce, 2004] David Pearce. Simplifying logic programs under answer set semantics. In *ICLP'04*, pages 210–224, 2004.

[Rintanen, 2000] Jussi Rintanen. An iterative algorithm for synthesizing invariants. In *AAAI'00*, pages 806–811, 2000.

[Saitta and Zucker, 2013] Lorenza Saitta and Jean-Daniel Zucker. Abstraction in artificial intelligence. In *Abstraction in Artificial Intelligence and Complex Systems*, pages 49–63. Springer, 2013.

[Saribatur and Eiter, 2021] Zeynep Saribatur and Thomas Eiter. Omission-based abstraction for answer set programs. *Theory and Practice of Logic Programming*, 21(2):145–195, 2021.

[Saribatur and Woltran, 2023] Zeynep G. Saribatur and Stefan Woltran. Foundations for Projecting Away the Irrelevant in ASP Programs. In *KR'23*, pages 614–624. IJCAI Organization, 2023.

[Saribatur and Woltran, 2024] Zeynep G. Saribatur and Stefan Woltran. A unified view on forgetting and strong equivalence notions in answer set programming. In *AAAI'24*, pages 10687–10695. AAAI Press, 2024.

[Saribatur *et al.*, 2021] Zeynep G. Saribatur, Thomas Eiter, and Peter Schüller. Abstraction for non-ground answer set programs. *Artificial Intelligence*, 300:103563, 2021.

[Saribatur *et al.*, 2024] Zeynep G. Saribatur, Matthias Knorr, Ricardo Gonçalves, and João Leite. On abstracting over the irrelevant in answer set programming. In *KR'24*, 2024.

[Sievers *et al.*, 2019] Silvan Sievers, Gabriele Röger, Martin Wehrle, and Michael Katz. Theoretical foundations for structural symmetries of lifted PDDL tasks. In *ICAPS'19*, volume 29, pages 446–454, 2019.

[Srivastava *et al.*, 2011] Siddharth Srivastava, Neil Immerman, and Shlomo Zilberstein. A new representation and associated algorithms for generalized planning. *Artificial Intelligence*, 175(2):615–647, 2011.

[Stockmeyer, 1976] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):1–22, 1976.

[Tarzariol *et al.*, 2023] Alice Tarzariol, Martin Gebser, Konstantin Schekotihin, and Mark Law. Learning to break symmetries for efficient optimization in answer set programming. In *AAAI'23*, pages 6541–6549. AAAI Press, 2023.

[Wrathall, 1976] Celia Wrathall. Complete Sets and the Polynomial-Time Hierarchy. *Theoretical Computer Science*, 3(1):23–33, 1976.

[Yu *et al.*, 2023] Haizi Yu, Igor Mineyev, and Lav R. Varshney. A group-theoretic approach to computational abstraction: Symmetry-driven hierarchical clustering. *Journal of Machine Learning Research*, 24(47):1–61, 2023.